# Auto-completion for Network Configurations

Ahsan Mahmood[*], Aaron Gember-Jacobson
*Colgate University*
{amahmood,agemberjacobson}@colgate.edu

## 1 Introduction

Most networks rely on distributed routing protocols to determine how traffic flows through the network. This requires configuring each device to run the appropriate protocols, communicate with other devices, and select the desired paths. The configurations are often complex, consisting of thousands of lines of low-level directives and dozens of symbolic references [2]. Consequently, configuration errors are common and the leading cause of network outages [5].

The prevalence of configuration errors stems from the rudimentary manner in which devices are typically configured. Most network devices feature a command line interface (CLI) for adding and removing individual lines of configuration. The only configuration assistance the CLI provides is tab-completion of keywords; this has limited value because keywords are listed alphabetically and the operator still has to search for the desired completion. To simplify common configuration tasks, many network management tools support the use of configuration templates: snippets of configuration that can be customized and inserted into the configurations of one or more devices [2]. However, these tools do not assist the operator in writing the templates or selecting the appropriate template(s) to use in a given configuration. More recently, network operators have begun to configure devices using higher-level, vendor-independent languages which are automatically compiled to the low-level device configurations [1]. But, these higher-level specifications can still contain dozens of symbolic references and be hundreds of lines long.

We thus propose a different approach that can serve to complement existing techniques for writing routing configurations by considering the problem of writing network configurations to be analogous to writing software code. Most configurations are written using vendor specific languages, that make use of rules and keywords similar to traditional programming languages. We envision an interactive system inspired by code completion engines [6, 4] that could be invoked by network operators as they write router configurations to offer them suggestions for what to put in next, or list the options available from the invocation point. Our long term goal is to expand this engine into a fully featured assistant for writing network configurations, one which can infer the type of router an operator is trying to configure and suggest relevant statements or even stanzas.

[*]Undergraduate student

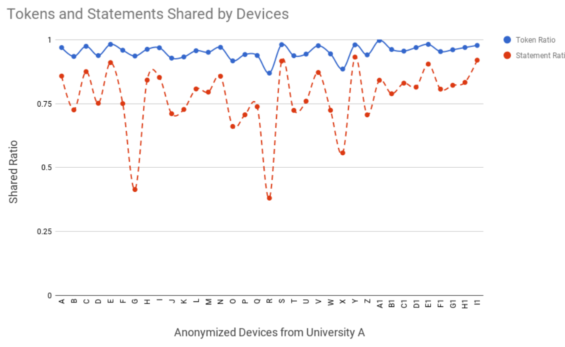

Figure 1: Token and statement similarity for Univ-A

Recent research on software systems has shown that codebases tend to contain regularities, much like natural languages [3]. This has motivated further research on using traditional Natural Language Processing techniques for code completion and token suggestion, resulting in fairly accurate models [3, 6]. We hypothesize a similar regularity for network configurations, especially since they tend to be homogeneous by design, reusing the same set of keywords/tokens. Our preliminary results show that using an off-the-shelf NLP algorithm with minor modifications, can give us accuracies as high as 95% for some configurations.

## 2 Configuration Similarity

Previous measurement students have found extensive use of templates in configurations from actual networks. Thus, we expect a network's configurations to share a common set of tokens and statements. To confirm this hypothesis, we split the configurations of a large university network (University A in Table 1) into tokens—each keyword and identifiers (e.g., interface names, VLAN numbers, IP prefixes, etc.) is considered a token. For each device, we measured the fraction of tokens and statements (i.e., lines of configuration) that existed in at least one other device's configuration.

As shown in Figure 1, almost all configurations were composed of the same set of unique tokens. The tokens that are not shared between configurations are primarily

| Univ. | No. of Configs | Total Lines | Avg Lines |
|---|---|---|---|
| A | 35 | 73K | 2.1K |
| B | 26 | 61K | 2.3K |
| C | 24 | 67K | 2.8K |

Table 1: Configurations used in our evaluation

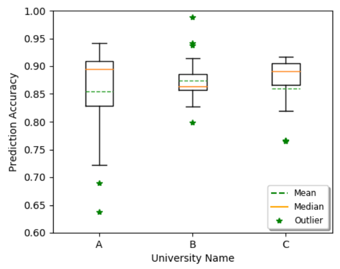

Figure 2: Prediction accuracy per-router per-network

IP prefixes. Moreover a large fraction of statements appeared in another device's configuration. These observations indicate that most of a device's configuration could be constructed from existing configurations.

## 3 NLP-based Configuration Completion

We leverage the regularity of configurations to design an intelligent configuration completion engine. According to Hindle et al. [3], regularities in texts can be easily exploited by natural language processing (NLP) techniques. Hence, we use n-grams, ranked based on likelihood, to complete the next token(s) in a configuration.

Prior to building the model, we employ a networking-specific optimization inspired by our observation that IP prefixes are often unique to devices (Section 2): we replace prefixes with generic `PREFIX` tokens. As we show below, this allows us to accurately predicate the next token in configuration statements involving prefixes.

**Preliminary Results.** We applied our framework to Cisco configurations of core, border, and distribution routers from three large university networks (Table 1). To test the accuracy of our model, we perform leave-one-out cross validation: one (set of) configuration(s) is used for testing and the remainder are used for training. We "rebuild" the test configuration(s) token-by-token by using our n-gram model to predict the next token based on the prior n-1 tokens; we do not predict across lines. A prediction is marked as successful when the actual next token in the configuration is within the top three results generated by the model.

Figure 2, shows the prediction accuracy for the routers in each network. Our approach achieves a high prediction accuracy (>85%) for the majority of routers. Without our placeholders optimization, this accuracy is 5% lower. We also analyzed the effects of training on more configurations in time and space. As shown in Figure 3, our framework does not require a long history of configurations to achieve reasonable accuracy. In contrast, training on more devices results in higher accuracies (Figure 4). However, training on more devices has diminishing returns, because additional devices play the same role as existing devices, and hence are very similar.

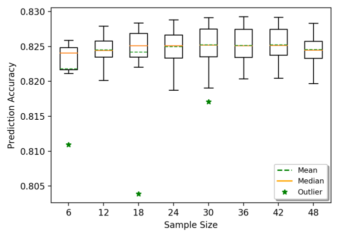

Figure 3: Impact of number of months for Univ-A

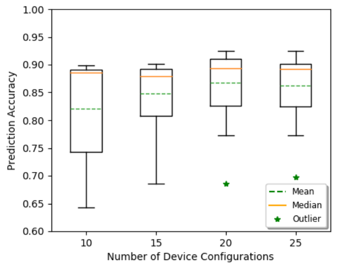

Figure 4: Impact of number of devices for Univ-A

## 4 Future Work

Our analyses help direct our attention towards areas of improvements for the model. The variance seen in our device analysis suggests that having different models for router of different "roles" could help improve prediction accuracies. Additionally, we plan on exploring the possibility of using larger n-grams to suggest complete statements. Lastly, we hope to evaluate our model against the current state of the art: tab-completion in CLIs on modern routers.

## References

[1] KEES: the coloclue network automation toolchain. `https://github.com/coloclue/kees`.

[2] T. Benson, A. Akella, and D. Maltz. Unraveling the complexity of network management. In *NSDI*, 2009.

[3] A. Hindle, E. T. Barr, M. Gabel, Z. Su, and P. T. Devanbu. On the naturalness of software. *Commun. ACM*, 59(5):122–131, 2016.

[4] JetBrains. Intellij autocompletion documentation. `https://goo.gl/1MrE8o`.

[5] Juniper Networks. What's behind network downtime? `https://www-935.ibm.com/services/au/gts/pdf/200249.pdf`.

[6] V. Raychev, M. Vechev, and E. Yahav. Code completion with statistical language models. In *PLDI*, 2014.